THE STORAGE ALLOCATION OF THE LINEAR PROGRAMMING CODE*

H. A. Judd
International Business Machines Corporation

P-907

July 26, 1956

*Prepared for The RAND Corporation Short Course in Computational Aspects of Linear Programming, Sept. 4–13, 1956.

CLEARINGHOUSE FOR FEDERAL SCIENTIFIC AND TECHNICAL INFORMATION CFSTI
DOCUMENT MANAGEMENT BRANCH 410.11

LIMITATIONS IN REPRODUCTION QUALITY

ACCESSION # AD 605066

☒ 1. WE REGRET THAT LEGIBILITY OF THIS DOCUMENT IS IN PART UNSATISFACTORY. REPRODUCTION HAS BEEN MADE FROM BEST AVAILABLE COPY.

☐ 2. A PORTION OF THE ORIGINAL DOCUMENT CONTAINS FINE DETAIL WHICH MAY MAKE READING OF PHOTOCOPY DIFFICULT.

☐ 3. THE ORIGINAL DOCUMENT CONTAINS COLOR, BUT DISTRIBUTION COPIES ARE AVAILABLE IN BLACK-AND-WHITE REPRODUCTION ONLY.

☐ 4. THE INITIAL DISTRIBUTION COPIES CONTAIN COLOR WHICH WILL BE SHOWN IN BLACK-AND-WHITE WHEN IT IS NECESSARY TO REPRINT.

☐ 5. LIMITED SUPPLY ON HAND: WHEN EXHAUSTED, DOCUMENT WILL BE AVAILABLE IN MICROFICHE ONLY.

☐ 6. LIMITED SUPPLY ON HAND: WHEN EXHAUSTED DOCUMENT WILL NOT BE AVAILABLE.

☐ 7. DOCUMENT IS AVAILABLE IN MICROFICHE ONLY.

☐ 8. DOCUMENT AVAILABLE ON LOAN FROM CFSTI (TT DOCUMENTS ONLY).

☐ 9.

PROCESSOR: GL

TSL-107-10 64

SUMMARY

The handling of storage assignment and subroutine interconnections for the 704 linear programming codes are discussed. The available storage of the machine is divided into two main parts -- code and data. Their layouts are fairly independent of one another and the programs include routines for automatic "housekeeping" during loading and running of a job. Flexibility is provided for systematic modifications in the programs or for machines with various storage facilities.

I have two reasons for discussing the storage allocation that we used in the coding of the Linear Programming problem. The foremost reason is to increase your understanding of the operation of the LP code, and secondly, I think it is an excellent solution of the storage allocation problem for any big general program. The 704 code for solving the Transportation problem, which I worked on in New York, solves the Transportation problem admirably, but it lacks the tidiness that a good storage layout provides.

To begin with, the LP code was designed for a 704 with 4096 words of magnetic core storage, one drum unit, five tape units, a card reader, a printer, and a punch. Only four tape units are necessary if no off-line printing is desired. The code has already been modified to take advantage of the additional 4096 words of core storage which has been added to some of the 704's now using the LP code. A more extensive modification is planned for the 32,000 word core storage unit.

The assembly of input data is, at present, completely separated both logically and physically from the LP main code. The data assembly program is made up of many subroutines to accommodate the different options which are allowed. For example, the data may be loaded on a tape via the peripheral card-to-tape equipment instead of the card reader. The Data Assembly code contains the necessary subroutines to accept the data either way. The layout is somewhat similar in structure to the main code since we wanted to use the same

data and constants regions. Since the Data Assembly code did not fit into the assigned space in high-speed storage, we broke it in half storing the second half on a drum. After the first half is executed, the second half is read in from drums overwriting the first half. Control is then returned to the beginning of the code and the second half is executed. This type of sleight of hand is very common where space is short and programs are long. It works best, however, when the program need be executed only once before it is destroyed, as in this case.

For the main code, high-speed storage is divided up as follows during the execution of the code:

0-- 15	COMMON	16
16-- 391	MASTER CONTROL ROUTINES (MCR)	376
392-- 487	UNIVERSAL SUBROUTINES (DPFADD, DPFMUL, DISTRB)	96
488-- 703	SUBROUTINES	216
704-- 767	UNIVERSAL CONSTANTS	64
768--	H-REGION	M+1
--	V-REGION	2M+2
--	W-REGION	2M+2
2048--4096	T-REGION	2048

The COMMON region is used for erasable storage by any MCR or subroutine at any time. The MCR region is used by the three different Master Control Routines which we have at the present time. Only one of these may be operating at a given time. In normal operation, the COMMON MCR is used until an optimal solution is reached, or until the operator decides to stop and invert the present basis. The INVERSION MCR must then be read into the MCR region from binary cards to perform the inversion. After the inversion is completed, the COMMON

MCR would be read in again from binary cards and the computation could proceed. This manual switching of the MCR codes may be eliminated by saving the MCR's internally, but at the time these MCR's are loaded, it may also be desirable to load part or all of the data in again since the restarts are handled in this way. In the near future some changes may be made in regard to the way restarts are handled so the MCR's will be saved internally for more automatic changing.

There are three universal subroutines, namely, Double Precision Floating Point Add, Double Precision Floating Point Multiply, and a subroutine called the Distributor. The functions of the first two are adequately explained by their titles. They are always available for use by any of the subroutines (hence, they might be more aptly described as sub-subroutines). The Distributor is a very short subroutine which contains a table of the number of instructions and the drum address of the first instruction of each subroutine in the code (with the exception of the Universal Subroutines). At present, there are 14 subroutines which may be used by the master codes. These subroutines do a great deal more than the ordinary macro-instruction type subroutines. The pricing operation, for example, is a single subroutine. When the MCR wants to execute this subroutine, it is only necessary to execute the two instructions

```
CLA PRICE
TSX DISTRB, C
```

The first instruction places the code number corresponding to the desired subroutine in the accumulator register so the Distributor can select the proper drum address from the table. The second instruction places the 2's complement of its own location in index register C and transfers control to the Distributor. First, the Distributor saves the contents of index register C in a standard location. Then the drum address of the first instruction and total number of instructions in the subroutine are taken from the table. The Distributor reads the subroutine from the drum into the standard subroutine region and transfers control to the first instruction of that region (488). The subroutine is then executed. Upon completion of the subroutine, index register C is reloaded from the standard location with the 2's complement of the location of the TSX instruction in the MCR. A TRA 1,C is executed which returns control to the instruction following the TSX instruction in the MCR. In the 704, the effective address of a tagged TRA instruction is computed by subtracting the contents of the indicated index register from the stated address in the instruction. The index registers do not have an associated sign, hence, the subtraction is done by taking the 2's complement of the number in the index register and adding it to the stated address. The 2's complement of the 2's complement is the number itself. In the example above, we added one to the original location and thus returned control to the instruction following the TSX

instruction in the MCR. If you are not confused by now, the rest will be easy.

Since it is possible to move 10,000 instructions per second between drum and high-speed storage, very little time is used by keeping all of the subroutines on drum and calling them in fresh each time they are to be executed. Thus, all of the "heavy" work was relegated to subroutines while decision making and control remained to the MCR's. Thus, to call a subroutine off the drum and execute it requires only two instructions in the MCR, the Distributor routine, and a very short amount of time. Each of the subroutines must necessarily make certain assumptions concerning the disposition of the data which it uses. It is the prime function of the MCR's to control the various activities and flow of data using the subroutines like the powerful tools they are.

In the space allotted, you will note that a subroutine cannot exceed 216 words in length. Later on, I will show how we can fudge this number up a bit, but first I want to point out the storage for constants and data. The Universal Constants contain the numbers 0, 1, 2, and other commonly used integers. They also include all of the standard locations for parameters to indicate which phase, which stage, which caption the print program will use, etc. Some of these constants are fixed by the Data Assembly program to define the storage layout for the particular problem being run such as M+1, 2M+2, VORIGN, WORIGN, etc.

The H-REGION always contains M+1 words and is used to keep the names of the current activities which constitute the basis. The V and W regions are 2M+2 words in length and each may contain a double-precision vector. The solution vector, Beta, is normally kept in V-region from where it is used. The W-region is used for working storage in generating $\alpha^{i}_{s(T)}$ and $\pi_{i}^{(T)}$ while working with the transformation vectors in T-region. It is used for several purposes, all of which require an expanded double precision vector. The transformation vectors, Eta_{t}, are kept in condensed form in T-region when they are being used. At other times, they are stored on drum 3, or transferred to tape (end-of-stage procedure) if they exceed the capacity of T-region. The only difference between the code for the 4096 and the 8192 word core storage machines is that T-region is expanded to half of 2048 plus 4096, that is from 2048 to 3072. Drum 3 is not used at present in the 8192 word machine. In place of using the drum for temporary storage for the Eta vectors, the high-speed storage following T-region is used. This change has caused a considerable increase in speed over the 4096 word machine. The T-region is also used for temporarily holding the matrix, or as much of it as possible, when it is used.

The tapes are used for permanent storage of the matrix (tape number 5), for permanent storage of the transformation vectors (tapes numbered 2, 3, and 4), and tape 6 is used as output by the print program if peripheral printing is desired.

Three tapes are needed for the transformation vectors for reliability and because they are used in both a forward and a backward direction. Tape 2 contains the vectors in the order in which they were generated providing, of course, that they exceed the capacity of T-region. Tapes 3 and 4 are used alternately to contain the transformation vectors in backward direction with respect to records only. That is, at the first end-of-stage, the record is written on tapes 2 and 3. At the second end-of-stage, the new record is added to tape 2 and written on tape 4. Then the contents of tape 3 are written on tape 4 following the new record. Tapes 3 and 4 are alternated in this way as long as necessary.

The drum layout is straightforward and flexible enough so that it can be changed around if desired. There are four logical drums in a drum unit of the 704 with 2048 words each. The first logical drum is selected by the drum load button sequence which we use extensively for restart procedures. Thus, the program labeled DRBOOT, for drum bootstrap, has to be placed at the beginning of drum 1.

The last 2M+2 words of drum 1 have been arbitrarily chosen to keep b^1, the original right-hand side. All of the other data including the current MCR and constants are saved on drum 1 for the automatic restart. Drum 2 and the rest of drum 1 are used for sub-MCR's and all of the subroutines. Drum 3 is used for storing transformation vectors when T-region is being used for other purposes except when using an 8192 word

machine. Drum 4 is used for other vectors generated during the runs which must be temporarily saved during the parametric programming.

Now I would like to show the way in which we are able to cheat on the rigid storage layout of both the MCR region and the subroutine region. The inversion MCR exceeds the 376 word limitation on the size of the MCR's. Here again, we use the sleight of hand technique of loading a part of the inversion MCR as a subroutine with a code of zero. The main part of the inversion MCR calls in this sub-MCR via the distributor. This sub-MCR knows a priori where the MCR gets stored on drum 2 so that it calls it in to the MCR region overwriting itself after it has finished its function.

The program which loads the subroutines, the sub-MCR's, and the current MCR is known as the executive loader (EXECLD). This is a standard binary loading program which has been modified for our special needs. The standard binary loader reads cards and stores the words in storage getting the number of words on each card and the loading address of the first instruction from the left word in the 9's row on the card. A hash check sum of the card is in the right word in the 9's row. Before it stores the instructions away, it tests the word count to see if it is zero. When it finds a card with a count of zero, it uses this indication to stop loading and transfers control to the location indicated in the address part of the left word in the 9's row.

The executive loader uses the same indication, a count of zero, and applies a detailed examination of the right word in the 9's row. For clarification, a card with a word count of zero is called a TITLE card instead of a transfer card in our write-ups. The number in the 9's right row of the title card indicates whether a subroutine, sub-MCR or MCR has just been loaded.

While the executive loader is loading these sub-MCR's and subroutines, it stores them away first on drum 2 and then on drum 1 when drum 2 is full. It keeps track of the first and last loading address of each code being loaded between successive title cards. Thus it computes the exact number of instructions in the routine which it adds to a counter to keep track of the space left on the drum which it is loading. Prior to loading each routine it places a return jump (TRA 1, C) in the first location of the MCR region. This is automatically destroyed by loading MCR's and also by some subroutines since we use another device for increasing the total length of subroutines. This is to put a preface to some subroutines in the MCR region and execute this set of code before the executive loader tucks the subroutine away on a drum. The preface is used to preset those addresses in each subroutine which depend on the particular problem the code is working on. You may recall that M is variable for each problem and determines the limits of H, V, and W regions. Thus the executive loader always transfers control to the first

instruction of the MCR where the preface is executed, if there is one, and control returns to the executive loader after the subroutine is initialized. Then the executive loader ignores the preface in the MCR region and tucks the condensed subroutine away on the drum, places the number of instructions and first drum location in the table of the Distributor routine and proceeds to load the next subroutine or MCR. Thus, the order of loading is determined to be:

BASE routines (DRBOOT, Universal sub-routines, EXECLD, the CONSTANTS)
SUBROUTINES
SUB-MCR's
MCR
DATA

The inherent beauty in this loading scheme is that the programmer does not have to know how many instructions are in each subroutine, sub-MCR, or MCR so long as the region boundaries are not violated. Again, we have allowed the 704 to do the housework of keeping track of the number of instructions in each subroutine, store them away on drums efficiently, and bring them back again when they are called for by the MCR.

The print subroutine exceeds the subroutine storage so badly that we have a pseudo-subroutine in its place and the real print program is loaded like a sub-MCR. The pseudo-subroutine gets called in and it in turn calls the real print program into T region since that is the only unoccupied space available which is large enough to accommodate the print program.

In conclusion, let me point out that data storage is a function of the number of equations in the system the code is working on. The storage of the LP code is independent of the problem and parts can be reassembled and included as they are needed. If we had a 704 with 32,000 words of core storage and no drum, we would use a block of core storage for the main subroutine storage. Any change like this affects only two routines, the executive loader and the distributor. If we get drums and 32,000 words of core storage, we would still keep the subroutines on drum and use all additional high-speed storage for transformation vectors.